# Conservative conversion between LaTeX and TeXmacs*

by Joris van der Hoeven[a], François Poulain[b]

LIX, CNRS
École polytechnique
91128 Palaiseau Cedex
France

a. *Email:* vdhoeven@lix.polytechnique.fr
b. *Email:* fpoulain@lix.polytechnique.fr

*February 27, 2014*

**Abstract**

Back and forth converters between two document formats are said to be conservative if the following holds: given a source document $D$, its conversion $D'$, a locally modified version $M'$ of $D'$ and the back conversion $M$ of $M'$, the document $M$ is a locally modified version of $D$. We will describe mechanisms for the implementation of such converters, with the LaTeX and TeXmacs formats as our guiding example.

**Keywords:** Conservative document conversion, LaTeX conversion, TeXmacs, mathematical editing

**A.M.S. subject classification:** 68U15, 68U35, 68N99

## 1 Introduction

The TeXmacs project [7, 10] aims at creating a free scientific office suite with an integrated structured mathematical text editor [8], tools for graphical drawings and presentations, a spreadsheed, interfaces to various computer algebra systems, and so on. Although TeXmacs aims at a typesetting quality which is at least as good as TeX and LaTeX [14, 15], the system is not based on these latter systems. In particular, a new incremental typesetting engine was implemented from scratch, which allows documents to be edited in a wysiwyg and thereby more user friendly manner. This design choice made it also possible or at least easier to use TeXmacs as an interface for external systems, or as an editor for technical drawings. However, unlike existing graphical front-ends for LaTeX such as LyX [2] or Scientific Workplace [23], native compatibility with LaTeX is not ensured.

*. This work has been supported by the Digiteo 2009-36HD grant and Région Ile-de-France.

Since LaTeX is still the standard for scientific publications in areas such as mathematics, physics and computer science, good compatibility between TeXmacs and LaTeX is a major issue. Several use cases are possible in this respect. For the publication of papers, good converters from TeXmacs to LaTeX are a prerequisite. New TeXmacs users also would like to import their old LaTeX papers into TeXmacs. The most complex types of conversion arise when a TeXmacs user collaborates with a person who refuses to use anything else but LaTeX. In that case, there is a need for lossless converters between both formats.

Unfortunately, TeX and LaTeX do not really provide a data format, but rather a programming language. Furthermore, unlike most other existing programming languages, the TeX system does not provide a formal grammar for the set of parsable documents. Moreover, the TeX syntax can be self-modified at run-time and many basic TeX/LaTeX capabilities are build upon syntactic tricks. This makes it extremely hard (or even impossible in practice) to design lossless converters between LaTeX and essentially different formats. Current software for conversions from LaTeX [1, 2, 4, 5, 6, 11, 13, 16, 17, 18, 19, 20, 21] therefore involves a lot of heuristics; we refer to section 3 for a quick survey of existing approaches.

Nevertheless, even if we accept that the conversion problem is hard in general, we would like our heuristic converters to address some important practical use cases. For instance, assume that Alice writes a LaTeX document and sends it to her colleague Bob. Now Bob makes a minor correction in the LaTeX document using TeXmacs and sends it back to Alice. Then Alice would like to recover her original LaTeX document except for the minor change made by Bob (which might possibly be exported in the wrong way). Converters between LaTeX and TeXmacs which admit this property will be called *conservative*.

There are several approaches to the implementation of conservative converters. First of all, we might re-implement our converters from scratch while taking into account the additional requirement. However, it took a lot of work and effort to develop the existing heuristic converters, so this option is not particularly nice from the implementers' perspective. Another idea would be to "hack" some parts of the existing code and turn it into something more conservative. Nevertheless, the existing converters are extremely complex; in particular, they cover numerous kinds of irregularities inside LaTeX. Adding an additional layer of complexification might easily break existing mechanisms.

Therefore, we want to regard the current converters between LaTeX and TeXmacs as black boxes. The aim of this paper is to describe conservative converters on top of these black boxes, which are currently under development. In particular, we guarantee that failure of the conservative converters only occurs in cases of failure of the original converters. Although our techniques were only tested for conversions between TeXmacs and LaTeX, it is likely that they can be used for other formats as well.

The first major ingredient for our converters (see section 4) is to generate, along with the main conversion, a correspondence between well chosen parts of the source document and their conversions in the target document. In particular,

we want to track the image of each paragraph in the source document. Ideally speaking, assuming that we have a nice parse tree for the source document, we want to track the images of all subtrees. In order to construct such correspondences, we will add markers to the source document, convert the marked source document, and finally remove the markers from the obtained marked target document. Optionally, we may want to verify that the addition and removal of markers does not disrupt the conversion process and that we obtain the same final document as in the case of a plain conversion.

For the conservative conversion itself, there are several approaches, which will be described in section 5. Our current implementations are mainly based on the "naive" approach from sections 5.1 and 5.2, which directly attempts to substitute unaltered document fragments by their original sources, when performing a backconversion of a modified converted document. We are also experimenting with more elaborate approaches for which modifications are regarded as patches and where the idea is to lift the conversion process to such patches.

## 2 The LaTeX and TeXmacs document formats

### 2.1 The TeX and LaTeX formats

As we mentioned in the introduction, TeX [14] is really a programming language rather than a data format. The main emphasis of TeX is on presentation and typesetting quality. TeX programs are strings with a highly customizable syntax; the set of TeX primitives can also easily be extended with user defined macros. We recall that LaTeX [15] is a set of macros, built upon TeX. It is intended to provide an additional layer of structural markup, thereby allowing for a limited degree of presentation/content separation. It inherits both advantages and disadvantages from the TeX system.

One of the major disadvantages of TeX and LaTeX is that it is hard or even impossible to write reliable converters to other formats. For instance, in a recent case study on existing LaTeX to MathML converters [24], it turned out that the success rates of these converters varied between 2% and 54%, when applied to a large document base downloaded from ArXiv, and looking at the mathematical formulas only. There are numerous reasons for this poor performance. Let us briefly mention the main ones:

*Lack of formal syntax.* TeX documents do not comply to any well defined syntax, since the syntax can be modified at run time. For instance, it is rather easy to typeset an HTML snippet such as `<a href="#ex1">example</a>` in the intended way using suitable TeX code. And, even though LaTeX widely advertises structured syntax, many LaTeX packages actually tweak TeX syntax in order to include external material (such as source code, algorithms, graphics) or to provide syntactic sugar, allowing to write code such as `foo!42!bar!!+++` (xcolor package) or `?[l]f*_ij^kl?` (tensid package). As a consequence, all existing parsers (except TeX-based parsers) are only able to correctly parse subsets of valid TeX documents.

*Lack of semantics.* The semantics of certain LaTeX language constructs (such as `\csname` or `\expandafter`) may be hard to specify, and even harder to translate into other data formats. Indeed, it's quite usual to see code such as `\def\b{\begin}` or `\def\wd{\widehat}` in real life LaTeX documents, which is completely meaningless when translated into a non-macro language. LaTeX also dramatically lacks of "public interfaces". So much so that many customisations are done using side effects, by redefining internal macros. So again, all existing translators are only able to translate subsets of valid TeX documents.

*Lack of orthogonality.* LaTeX, as a document format, dramatically lacks of orthogonality. Numerous packages are mutually incompatible and numerous document classes are providing different macro naming conventions, making code non portable between classes (*e.g.* the declaration `\newtheorem{thm}{Theorem}` may be used with the article class, but is forbidden with elsart). Numerous macros suffer from arbitrary limitations (*e.g.* a paragraph cannot end inside a `\texttt` argument; the nesting level in lists is limited to 4, etc.). Each document class define its very own scheme for defining metadata and title pages. The characters <, > and | behave very differently depending on the preamble. Etc. All these irregularities make difficult to build generically valid LaTeX documents.

## 2.2 The TeXmacs format

The TeXmacs document format is based on *trees*: all TeXmacs documents or document fragments should be thought of as trees [9]. Of course, such trees can be serialized as strings in different ways: as XML documents, as Scheme expressions (S-expressions), etc. In what follows, we will represent TeXmacs trees using S-expressions. Figure 1 shows an example of a TeXmacs snippet.

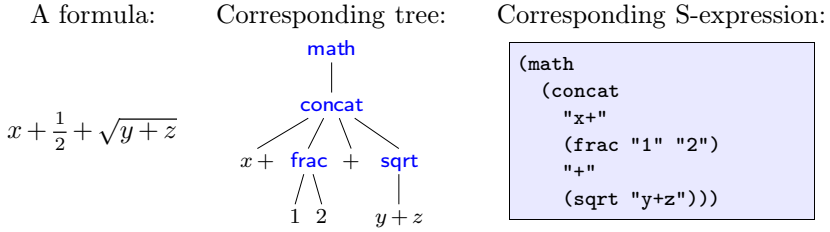



**Figure 1.** An insight of the TeXmacs format.

In the example, the `concat` tag stands for "horizontal concatenation" and its children expect "*inline* content". Another important primitive which will be used in subsequent examples is `document`; this tag is used for "vertical concatenation" and it provides "*block* content".

# 3 Traditional converters for LaTeX

## 3.1 Existing approaches

Not so surprisingly, the interoperability with LaTeX is an asymmetric problem. Conversion to LaTeX is considered the easier problem, since we are free to restrict ourselves to a subset of LaTeX. However, even this problem is quite non trivial, due to lack of orthogonality and arbitrary restrictions inside LaTeX (see section 2.1). Nevertheless, conversion from LaTeX to other formats is indeed the most difficult problem, usually. Although the techniques described in this paper work both ways, we will therefore use this harder direction for our examples.

The main obstacle when converting LaTeX documents to other formats is that arbitrary LaTeX documents are hard to parse. One way to bypass this problem is to let TeX do the parsing for you. Some existing converters which use this approach are TeX4ht, Tralics, LXir, Hermes and LaTeXml [1, 4, 5, 13, 18]. The underlying principle is to build an overloaded DVI document [3, 22] with additional markup which delimits the scope of a set of interesting macros. Then, via an external program, it is possible to rebuild a tree reflecting the structure of the original document. This alternative has the very benefit of exploiting TeX parser without rewriting it, but it has also major concerns:

1. It supposes that you installed a complete TeX distribution (usually large in size), whose version is compatible with the one required by the source.
2. It only works for complete documents, and not for code snippets.
3. User defined macros are expanded and thereby lose their semantics.

Concerning the last point, we agree with [12]: "macro expanding then translating is best suited to display and does not provide converted documents suitable for further use", so this strategy does not fit our needs. Also, even if we could preprocess sources in order to protect user defined markup, the two first concerns are too restrictive for our project.

The remaining strategy is to parse ourselves the LaTeX source files. This approach has been followed by software such as LyX, HeVeA, PlasTeX, Pandoc, Latex2html, Perl LaTeX::Parser, HyperLaTeX, TtH, etc. [2, 6, 11, 16, 17, 19, 20, 21] Once the document parsed, a tree is generated and is transformed in order to comply with the target format definition.

## 3.2 Traditional conversion from LaTeX to TeXmacs

The existing LaTeX to TeXmacs converter uses a custom TeX/LaTeX parser, with the ability to conserve the semantics of user defined macros. Many filters are applied to the parsed LaTeX document in order to get a suitable TeXmacs tree. Some example transformations which are applied are the following:

- expansion of "dangerous" macros, such as `\def\b{\begin}`;
- cutting the source file into small pieces, thereby reducing the risk of serious parse errors due to (text/math) mode changes;

- many local and global tree rewritings (whitespace handling, basic normalizations and simplifications, renaming nodes, reordering children, migrating label definitions, extracting metadata, etc.);
- adding semantics to mathematical content (disambiguation of operators and whitespace, matching delimiters, etc.);
- (optionally) compile the LaTeX document with the preview package in order to import hard-to-parse document fragments as pictures.

Due to the combined complexity of these transformations, we cannot make certain useful assumptions on the behavior of our converter, especially:

**(non) Linearity.** The conversion of a concatenation of snippets might not be the concatenation of the conversion of the snippets.

**(non) Locality.** Local changes inside snippets may result in global changes.

**(non) Time invariance.** Two conversions of the same document at different times might result in different results. This is for instance due to time stamping by external tools involved in the conversion of pictures.

### 3.3 Traditional conversion from TeX$_{\text{MACS}}$ to LaTeX

The converter from TeX$_{\text{MACS}}$ to LaTeX does not have to cope with the LaTeX parsing problem. Nevertheless, arbitrary restrictions inside LaTeX and the general lack of orthogonality make the production of high quality LaTeX documents (including preserved semantics of user defined macros) harder than it seems. The LaTeX export process is based on the following general ideas:

- converting the document body tree into a LaTeX syntax tree;
- building the preamble by tracking dependencies and taking into account all document style options;
- writing the preamble and the body.

Again, and for similar reasons as above, useful properties such as linearity, locality and time invariance cannot be guaranteed.

## 4 Correspondence between subdocuments

### 4.1 Basic principles

One important prerequisite for conservative converters is our ability to maintain the correspondence between parts of the source document and their images in the target document. When writing conservative converters from scratch, this correspondence can be ensured by design. In our setting, since traditional converters are already written, we consider them provided as if they were black boxes, independant from our conservative converters. Then, the key idea in order to maintain the correspondence between source and target is to add markers to the source document before doing the conversion. More precisely, we extend the source and target formats with one new special tag for marking text, say `marker`. Two design choices have now to be made.

First of all, the `marker` tag can either be punctual or scoped. In the first case, the tag takes a unique identifier as its single argument. In the second case, the tag takes the marked text as its additional second argument (for large multi-paragraph fragments of LaTeX, this requires `marker` to be an environment). For unparsed LaTeX source documents, substrings which correspond to logical subexpressions of the parse tree can be hard to determine. Consequently, punctual markers are most adequate for the initial conversions from LaTeX to TeXmacs. Nevertheless, using a post treatment, we will show in section 4.2 below that punctual markers may be grouped by pairs into scoped markers, while taking advantage of the tree structure of the generated TeXmacs document. For conversions from TeXmacs to LaTeX, we may directly work with scoped markers.

Secondly, we have to decide on the granularity of our conservative converters: the more markers we use, the more locality will be preserved for small changes. However, not all subtrees of the parse tree of the source document necessarily give rise to valid subexpressions of the target documents. Moreover, in the case of conversions from LaTeX to TeXmacs, we have no direct access to the parse tree of the source document, so appropriate places for markers have to be chosen with care. At least, we need to mark all paragraphs. More sophisticated implementations will also mark macro definitions in the preamble, cells of tables, and several well chosen commands or environments.

**Example 1.** In Figure 2, we have shown a simple example of added punctual markers inside a LaTeX source document. The granularity is slightly better than a purely paragraph based marking with which the markers 4 until 9 would be suppressed. Nevertheless, an even better granularity could for instance be obtained by putting markers around the `b` and `c` of the fraction. As a post treatment we typically group the punctual markers by pairs into scoped markers. In this case, the pairs are $(1, 2)$, $(3, 10)$, $(4, 7)$, $(5, 6)$, $(8, 9)$ and $(11, 12)$.

```
First paragraph.                \marker{1}First paragraph.\marker{2}

%Some comments                  %Some comments
\begin{remark}                  \marker{3}\begin{remark}
 Some mathematics                 \marker{4}Some mathematics
 \[ a+\frac{b}{c}. \]  ⇝          \[ \marker{5}a+\frac{b}{c}.\marker{6} \]\marker{7}

 More text.                       \marker{8}More text.\marker{9}
\end{remark}                    \end{remark}\marker{10}

Last paragraph.                 \marker{11}Last paragraph.\marker{12}
```

**Figure 2.** Rudimentary example of a marked LaTeX source document body.

### 4.2 Grouping punctual markers by pairs

In order to transform matching pairs of punctual markers inside LaTeX source documents into scoped markers, we will make use of the tree structure of the marked TeXmacs target document obtained after conversion.

Roughly speaking, for every subtree wich starts with marker $i$ and ends with marker $j$, we declare that $(i, j)$ forms a matching pair. More precise implementations should carefully take into account the special nature of certain tags such as `concat` and `document`. For instance, we may use the following algorithm:

- any `concat` tag or `document` tag starting and finishing by a marker is replaced by the corresponding pair;
- any child of a `concat` tag or a `document` tag which is framed by two markers is replaced by the corresponding pair;
- any `concat` tag or `document` tag with only one child which is a pair is replaced by the pair;
- any remaining marker is removed.

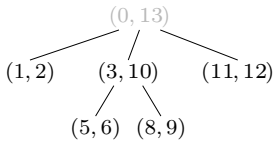
**Example 2.** For the simple document from Example 1, the TEXMACS conversion of the marked document would be as follows:

```
(document
  (concat (marker "1") "First paragraph." (marker "2"))
  (marker "3")
  (remark
    (document
      (concat (marker "4") "Some mathematics")
      (equation*
        (document (concat (marker "5") "a+" (frac "b" "c") "." (marker "6"))))
      (marker "7")
      (concat (marker "8") "More text." (marker "9"))))
  (marker "10")
  (concat (marker "11") "Last paragraph." (marker "12")))
```

The pairs (1, 2), (5, 6), (8, 9) and (11, 12) are detected as matching pairs of markers of inline content, whereas the pair $(3, 10)$ is detected as matching pair of block content. The pair (4,7) does not match, but the matching algorithm can be further tweaked to better handle this kind of situations.

At a second stage, we may organize the matching pairs into a dag: a pair $(i, j)$ will be a descendent of a pair $(p, q)$ whenever the source string corresponding to $(i, j)$ is included in the source string corresponding to $(p, q)$. We will say that the dag is *well formed* if it is actually a tree and whenever the source strings corresponding to two different children of the same node never intersect.

**Example 3.** For our example document, and adding an implicit pair $(0, 13)$ for the root, the dag of matching pairs is a well formed tree:



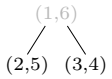
**Example 4.** Certain tags, such as `figure`, may produce badly formed trees:

```
\begin{figure}
\marker{2}\caption{\marker{3}Legend\marker{4}}
Some text.\marker{5}
\end{figure}
```

□

```
(big-figure
  (concat
    (marker "2")
    "Some text."
    (marker "5"))
  (concat
    (marker "3")
    "Legend"
    (marker "4")))
```

□




In unfavourable cases, such as Example 4, the dag of matching pairs does not yield a well formed tree. In such cases, we keep removing offending matching pairs until we do obtain a well formed tree. At the end of this process, we are ensured that marked subdocuments do not overlap in non structured ways, i.e. other than *via* inclusion. Consequently, it is possible to transform the document with punctual markers into a document with scoped markers.

**Example 5.** Applying the final transformation to Example 2, we obtain the document below. Of course, the names of our scoped identifiers can be replaced by simple numbers.

```
(document
  (marker "1:2" "First paragraph.")
  (marker "3:10"
    (remark
      (document
        "Some mathematics"
        (equation*
          (document (marker "5:6" (concat "a+" (frac "b" "c") "."))))))
      (marker "8:9" "More text.")))
  (marker "11:12" "Last paragraph." ))
```

### 4.3 Detection and reparation of irregularities

In most cases, removal of the markers from the conversion of a marked source document yields the same result as the direct conversion of the source document, modulo some normalization, such as merging and removing `concat` tags. However, this is not always the case. Let us give a few examples of how this basic principle may fail.

**Example 6.** When putting markers inside certain special tags, such as `verbatim`, the markers may not be converted as expected:

```
\begin{verbatim}
Some text.
\end{verbatim}
```

⇝

```
\begin{verbatim}
\marker{1}Some text.\marker{2}
\end{verbatim}
```

↕

```
(verbatim
  "\marker{1}Some text.\marker{2}")
```

**Example 7.** The conversion of certain LaTeX documents may involve some restructuring which is incompatible with the marking algorithm. For instance, TeXmacs only allows for modified text properties of block content if the modified properties apply to a succession of entire paragraphs. LaTeX also allows emphasized text which starts in the middle of a paragraph, runs over several subsequent paragraphs, and then ends in the middle of another paragraph. When importing this kind of ill structured emphasized text, we therefore restructure the emphasized text into three separate parts. However, such transformations often interfere in unpredictable ways with the marking process.

In order to guarantee that our marking and unmarking mechanisms never deteriorate the quality of the converters, we may use the following simple procedure: convert the source document both with and without the marking/unmarking mechanism. If the results are the same, then return the result. Otherwise, keep locating those markers which are responsable for the differences and put them into a blacklist. Now keep repeating the same process, while no longer inserting the markers in the blacklist during the marking procedure. In the worst case, this process will put all markers on the blacklist, in which case conversion with and without the marking/unmarking mechanism coincide.

### 4.4 Conservative storage of documents

In order to allow for subsequent conservative back and forth conversions, the result of a conservative conversion should contain additional information on how the recover the original source file. More precisely, when converting a source document $D$ from format $A$ into format $B$, the target document will consist of a quadruple $(D', \bar{D}, \bar{D}', \lambda)$, where

- $D'$ is the target document of format $B$.
- $\bar{D}$ is the marked source document which gave rise to the marked version $\bar{D}'$ of the target document.
- $\lambda$ is a mapping which associates one of the two formats $A$ or $B$ to every identifier for a marked subdocument.

If our target format is TeXmacs, then we simply store $\bar{D}$, $\bar{D}'$ and $\lambda$ as attachments to TeXmacs files. If LaTeX is our target format, then, by default, we put $\bar{D}$, $\bar{D}'$ and $\lambda$ inside a comment at the end of the document. Alternatively, we may store $\bar{D}$, $\bar{D}'$ and $\lambda$ in a separate file, which will be consulted whenever we convert a modification of the target document back to $A$. The last strategy has the advantage that we do not clobber the converted file. However, one will only benefit from the conservative converters when the LaTeX reimportation is done on the same computer and *via* the same user account.

**Remark 8.** Instead of specifying $\lambda$ as a separate mapping, it also possible to suffix identifiers for marked subdocuments by a letter for the original format of the marked text.

**Example 9.** When importing the LaTeX source file from Examples 1 and 5 into TeXmacs, the mapping $\lambda$ will associate "LaTeX" to each of the identifiers `1:2`, `3:10`, `5:6`, `8:9`, `11:12`. If the LaTeX source file was a modified version of the result of exporting a TeXmacs file to LaTeX, then the mapping $\lambda$ will associate TeXmacs to every node of the tree, except for those nodes which correspond to substrings in which modifications took place.

# 5 Conservative conversion

## 5.1 The naive approach

Let us return to the main problem of conservative conversion. Alice has written a document $D$ in format $A$, converts it to format $B$ and gives the resulting document $D'$ to Bob. Bob makes some modifications and send a new version $M'$ back to Alice. How to convert the new version back to format $A$ while conserving as much as possible from the original version for the unmodified parts?

Let us first describe a naive solution to this problem. We will assume that the format $B$ is enriched with one new unary tag `invariant`, with an expression of format $A$ as its unique argument. The conversion from $B$ to $A$ of such a tag will be precisely this argument.

In the light of the section 4.4, Bob's new version contains a marked copy $\bar{D}$ of Alice's original version as well as its marked conversion $\bar{D}'$ to format $B$. Now for every subdocument $S'$ occurring in $M'$ which corresponds to a marked subdocument of $\bar{D}'$ (and which is maximal with this property), we replace $S'$ by an `invariant` tag which admits the subdocument $S$ in $D$ corresponding to $S'$ as its argument. We finally convert the obtained document from $B$ to $A$ using our slightly adapted black box converter.

**Example 10.** Assume that Bob adds two more dots to the paragraph `More text.` in Example 2. Then the subdocuments corresponding to the pairs of markers $(1, 2)$, $(5, 6)$ and $(11, 12)$ still occur in the new document. Since $(8, 9)$ corresponds to a subdocument of $(3, 10)$, we only declare the subdocuments corresponding to $(1, 2)$, $(5, 6)$ and $(11, 12)$ to remain invariant. More precisely, we perform the conversion

```
(document                                   First paragraph.
  (invariant "First paragraph.")
  (remark                                   \begin{remark}
    (document                                 Some mathematics
      (invariant "Some mathematics           \[ a+\frac{b}{c}. \]
\[ a+\frac{b}{c}. \]")                  ⇝
      "More text..."))                        More text...
  (invariant "Last paragraph."))            \end{remark}

                                            Last paragraph.
```

Notice the change of indentation and the disappearance of the comment.

### 5.2 Fine tuning of the naive approach

A few additional precautions are necessary in order to make the naive approach fully work. First of all, during the replacement procedure of subdocuments $S'$ of $M'$ by invarianted subdocuments $S$ of $D$, some subdocuments $S'$ of $M'$ might correspond to several marked subdocuments $S_1, ..., S_n$ of $D$. In that case, we first investigate some of the context in which $S'$ occurred, such as the first $k = 1, 2, ...$ marked subdocuments before and after $S'$. In many cases, there will be only one subdocument $S_i$ which will both correspond to $S'$ and its context. If such a preferred match $S_i$ cannot be found for $S'$, then we renounce replacing $S'$ by an invariant tag.

Secondly, the conversion algorithms are allowed to be context dependent. For instance, ordinary text and mathematical formulas are usually not converted in the same way. When replacing subdocuments $S'$ of $M'$ by invarianted subdocuments $S$ of $D$, we thus have to verify that the context of $S'$ in $M'$ is similar to the context of $S'$ in $D'$.

Some other improvements can be carried out in order to further improve the quality of naive conservative conversions. For instance, in Example 10, we observed that the comment before the remark is lost. This would not have been the case if Bob had only modified the last paragraph. It is a good practice to detect adjacent unchanged portions of text and keep the comments in the corresponding parts of the original source file, but it may be difficult to achieve.

Additional *ad hoc* techniques were used to solve others problems. For instance, certain editors automatically damage the line breaking of LaTeX source code. This issue has been addressed by normalizing whitespace before testing whether two subdocuments are identical.

### 5.3 Patch based conservative conversions

Another strategy towards conservative editing is to determine the changes between the conversion $S'$ of Alice's version and Bob's version $D'$ in the form of a "patch" $\pi'$, and then try to convert this patch $\pi'$ into a patch $\pi$ that can be applied to $S$. Before anything else, this requires us to fix formats $A^\sharp$ and $B^\sharp$ for the description of patches for the formats $A$ and $B$.

For instance, LaTeX documents are strings, so LaTeX patches could be sets of triples $(i, j, R)$, where $(i, j) \in \mathbb{N}^2$ corresponds to a substring of the source string and $R$ stands for a replacement string for this substring. This language might be further extended with pairs $((i_1, ..., i_n), \sigma) \in \mathbb{N}^n \times \mathsf{S}_n$, where $(i_1, ..., i_n)$ is an $n$-tuple of positions $0 \sqsubset i_1 < \cdots < i_n \sqsubset l$ of the source string (of length $l$) and $\sigma$ a permutation. The patch then applies the permutation $\sigma$ to the substrings $(0, i_1)$, $(i_1, i_2), ..., (i_n, l)$ of the source string.

Similar patches can be used for $\text{TEX}_{\text{MACS}}$ trees, by operating on the children of a node instead of the characters in a string. In $\text{TEX}_{\text{MACS}}$, we also implemented a few other types of elementary patches on trees for the insertion or removal of

new nodes and splitting or joining nodes. However, we have not yet used these more complex kind of patches in our conservative converters. In general, we notice that richer patch formats lead to more conservative converters, but also make implementations more complex.

Let us study the most important kind of patches $\pi'$ which simply replaces a subdocument $X'$ of $D'$ by $Y'$. If $X'$ is marked inside $\bar{D}'$, with $X$ as the corresponding source in $D$, then we take $\pi$ to be the replacement of $X$ by contextual conversion $Y$ of $Y'$ into format $A$. This contextual conversion of $Y'$ is obtained by performing a marked conversion of $M'$ into format $A$ and then look for the conversion of $Y'$ as a subdocument of $M'$.

**Example 11.** Assume again that Bob adds to more dots to the paragraph `More text.` in Example 2. Then $\pi'$ is the patch which replaces subtree `"More text."` by `"More text..."`. This subtree `"More text."` corresponds to the pair of markers $(8, 9)$ and to the unique substring `More text.` in the original source document. Consequently, $\pi$ will be the patch which replaces this substring by `More text...`, which leads to the conservative conversion

```
First paragraph.

%Some comments
\begin{remark}
 Some mathematics
 \[ a+\frac{b}{c}. \]

 More text...
\end{remark}

Last paragraph.
```

### 5.4 Fine tuning of the patch based approach

Several things have to be fine tuned for the patch based approach. Example 11 is particularly simple in the sense that the patch $\pi'$ which replaces `"More text."` by `"More text..."` replaces a subtree by another tree. More generally, we have to consider the case when a succession of children of a subtree is replaced by a sequence of trees. This occurs for instance when inserting or deleting a certain number of paragraphs. For special types of nodes (such as the TEXMACS `document` tag for successions of paragraphs), we know how the node behaves under conversions, and we can devise an *ad hoc* procedure for computing the patch $\pi$. In general however, we may have to replace $\pi'$ by a less fine grained patch which replaces the entire subtree by a new tree.

A similar situation arises when the patch $\pi'$ replaces as subdocument $X'$ of $D'$ which is not marked inside $\bar{D}'$, or when the subdocument $Y'$ of $M'$ does not lead to marked subdocument $D$ of the marked conversion of $M'$ into format $A$. In these cases as well, a simple solution again consists of replacing $\pi'$ by a less fine grained patch which replaces a larger subtree by another tree.

**Example 12.** In Example 2, assume that Bob replaces the numerator `b` of the fraction by `x`. This corresponds to a patch $\pi'$ which does not operate on a marked subtree of $\bar{D}'$. Nevertheless, the patch which replaces the marked subtree `(concat "a+" (frac "b" "c"))` by `(concat "a+" (frac "x" "c"))` does admit the required form.

### 5.5 The combined approach

The quality of conservative converters can be further enhanced by combining the naive and patch based approaches. Assume that $M'$ is obtained from $D'$ through the application of a set $\pi'_1, ..., \pi'_n$ of independent patches (i.e., acting on disjoint subdocuments of $D'$). If $n > 0$, then we will reduce the general "patch conversion" problem to a problem of the same kind but with a strictly smaller number of patches $n'$.

Consider a subdocument $S'$ of $D'$ with the following properties:

- The subdocument is marked inside $\bar{D}'$ and corresponds to the subdocument $S$ of $D$.
- At least one of the patches $\pi'_i$ applies to a part of $S'$.
- $S'$ admits no strictly smaller subdocuments satisfying the same properties.

Let $T'$ be the result of applying all relevant patches $\pi'_i$ to $S'$. Now apply the marked version of the naive conversion techniques from sections 5.1 and 5.2 to $M'$. This will yield a conservative contextual conversion $T$ of $T'$ into format $A$.

We now consider the new "source document" $D^*$ obtained from $D$ through the replacement of $S$ by $T$. Similarly, we consider the new "conversion" $D'^*$ obtained from $D'$ through the replacement of $S'$ by $T'$. These replacements admit marked versions which basically change nothing outside $S$ and $S'$ and remove all markers strictly inside $S$ and $S'$. This completes our reduction of the general "patch conversion" problem to one with strictly less patches (namely, all remaining patches which did not apply to $S'$). In practice, several non overlapping subdocuments $S'$ can be treated in a single iteration, so as to increase the efficiency.

## 6 Conclusion

Conservative converters should make collaborations easier between people who are working with different authoring tools. Although we only considered conversions between TeXmacs and LaTeX here, our methods should also be useful for other formats. We also notice that the approach generalizes in a straightforward way to the documents which are written using three or more different tools or formats.

The implementations inside TeXmacs have only started quite recently and some time will still be needed for testing and maturing. Nevertheless, our first experiences are encouraging. Currently, we are still struggling with conservative conversions of user defined macros and metadata such as title information.

Nevertheless, we are confident that a suitable solution can be worked out even for complex conversion challenges of this kind, by finetuning the grain of our marking algorithm, and through the progressive integration of the patch based approach.

## Bibliography


**[1]** Romeo Anghelache. Hermes. http://hermes.roua.org/, 2005.

**[2]** M. Ettrich et al. The LyX document processor. http://www.lyx.org, 1995.

**[3]** David Fuchs. The format of TEX's DVI files. *TUGboat*, 3(2):13–19, 1982.

**[4]** José Grimm. Tralics. http://www-sop.inria.fr/marelle/tralics/, 2003.

**[5]** E. Gurari. TeX4ht: LATEX and TEX for hypertext. http://www.tug.org/applications/tex4ht/mn.html, 2010.

**[6]** Sven Heinicke. LaTeX::Parser – Perl extension to parse LATEX files. http://search.cpan.org/~svenh/LaTeX-Parser-0.01/Parser.pm, 2000.

**[7]** J. van der Hoeven. GNU TEXmacs: A free, structured, wysiwyg and technical text editor. In Daniel Filipo, editor, *Le document au XXI-ième siècle*, volume 39–40, pages 39–50, Metz, 14–17 mai 2001. Actes du congrès GUTenberg.

**[8]** J. van der Hoeven. Towards semantic mathematical editing. Technical report, HAL, 2011. http://hal.archives-ouvertes.fr/hal-00569351, submitted to JSC.

**[9]** J. van der Hoeven. *GNU TEXmacs User Manual*, chapter 14: The TEXmacs format. HAL, 2013. http://hal.archives-ouvertes.fr/hal-00785535.

**[10]** J. van der Hoeven et al. GNU TEXmacs. http://www.texmacs.org, 1998.

**[11]** Ian Hutchinson. TtH, the TEX to html translator. http://hutchinson.belmont.ma.us/tth/, 1997.

**[12]** Rodionov I. and S. Watt. A TeX to MathML converter. http://www.orcca.on.ca/MathML/texmml/textomml.html.

**[13]** Jean-Paul Jorda and Xavier Trochu. LXir. http://www.lxir-latex.org/, 2007.

**[14]** D.E. Knuth. *The TeXbook*. Addison Wesley, 1984.

**[15]** L. Lamport. *LATEX, a document preparation system*. Addison Wesley, 1994.

**[16]** J. MacFarlane. Pandoc: a universal document converter. http://johnmacfarlane.net/pandoc/index.html, 2012.

**[17]** L. Maranget. HeVeA. http://para.inria.fr/~maranget/hevea/index.html, 2013.

**[18]** B. Miller. LATEXML: A LATEX to XML converter. http://dlmf.nist.gov/LaTeXML/, 2013.

**[19]** R. Moore. LATEX2HTML. http://www.latex2html.org, 2001.

**[20]** T. Sgouros. HyperLaTeX. http://hyperlatex.sourceforge.net, 2004.

**[21]** Kevin Smith. plasTEX. http://plastex.sourceforge.net/, 2008.

**[22]** M. Sofka. TEX to HTML translation via tagged DVI files. *TUGboat*, 19(2):214–222, 1998.

**[23]** MacKichan Software. Scientific workplace. http://www.mackichan.com/index.html?products/swp.html~mainFrame, 1998.

**[24]** H. Stamerjohanns, D. Ginev, C. David, D. Misev, V. Zamdzhiev, and M. Kohlhase. MathML-aware article conversion from LATEX, a comparison study. In P. Sojka, editor, *Towards Digital Mathematics Library, DML 2009 workshop*, pages 109–120, Masaryk University, Brno, 2009. http://kwarc.info/kohlhase/papers/dml09-conversion.pdf.